

# Compact Indexes for Flexible Top- $k$ Retrieval

Simon Gog and Matthias Petri

Department of Computing and Information Systems  
The University of Melbourne, Parkville VIC 3010, Australia

**Abstract.** We engineer a self-index based retrieval system capable of rank-safe evaluation of top- $k$  queries. The framework generalizes the GREEDY approach of Culpepper et al. (ESA 2010) to handle multi-term queries, including over phrases. We propose two techniques which significantly reduce the ranking time for a wide range of popular Information Retrieval (IR) relevance measures, such as  $\text{TF} \times \text{IDF}$  and BM25. First, we reorder elements in the document array according to document weight. Second, we introduce the repetition array, which generalizes Sadakane’s (JDA 2007) document frequency structure to document subsets. Combining document and repetition array, we achieve attractive functionality-space trade-offs. We provide an extensive evaluation of our system on terabyte-sized IR collections.

## 1 Introduction

Calculating the  $k$  most relevant documents for a multi-term query  $Q$  against a set of documents  $\mathcal{D}$  is a fundamental problem – the top- $k$  document retrieval problem – in Information Retrieval (IR). The relevance of a document  $d$  to  $Q$  is determined by evaluating a similarity function  $\mathcal{S}$  (e.g.  $\text{TF} \times \text{IDF}$  or BM25). Naive exhaustive processing evaluates  $\mathcal{S}$  for each document  $d$  in  $\mathcal{D}$  and generates a full list of scores. The top- $k$  documents in the list are then reported. Algorithms which guarantee production of the same top- $k$  results list as the exhaustive process are called *rank-safe*.

The *inverted index* is a highly-engineered data structure designed to solve this problem. The index stores, for each unique term in  $\mathcal{D}$ , the list of documents  $d_i$  containing that term. Queries are answered by processing the lists of all of the query terms. Advanced query processing schemes [2] process lists only partially while remaining rank-safe. However, additional work during construction time is required to avoid scoring non-relevant documents at query time. Techniques used to speed up query processing include sorting lists in decreasing score order, or pre-storing score upper bounds for sets of documents which can then safely be skipped during query processing. These pre-processing steps introduce a dependency between the similarity measure and the stored index. Changing the scoring function requires at least partial reconstruction of the underlying index, which in turn reduces the flexibility of the retrieval system.

Another family of retrieval systems is based on self-indexes [9]. These systems support functionality not easily provided by inverted indexes, such as efficient

phrase search, and direct text extraction. Systems capable of single-term top- $k$  queries have been proposed [10] and have proven to work well in practice [7]. Generalizing and extending these structures to support multi-term queries and more complex similarity functions is essential to the adaption of self-indexes in the context of IR. However, currently only simple heuristics which cannot provide rank-safe query processing exist [5].

*Our Contributions.* We propose, to the best of our knowledge, the first self-index based retrieval framework capable of rank-safe evaluation of top- $k$  queries. In addition to the functionality of self-indexes (such as text extraction and phrase queries) it can process multi-term queries using complex IR relevance measures on terabyte scale IR collections. It is based on a generalization of the GREEDY approach of Culpepper et al. [4]. We suggest two techniques to decrease the number of evaluated nodes in the GREEDY approach. The first is reordering of documents according to their length (or other suitable weight), the second is a new structure called the *repetition array*,  $R$ . The latter is derived from Sadakane’s [12] document frequency structure, and is used to calculate the document frequency for subsets of documents. We further show that it is sufficient to use only  $R$  and a subset of the document array if query terms, which can also be phrases, are length-restricted. Finally, we explore the properties of our proposal on two terabyte-scale IR collections. This is, to our knowledge, three orders of magnitudes larger than in previous self-index based experiments. Our source code and experimental setup is publicly available.

*Paper outline.* In Section 2 we introduce notation, a formal problem definition, and examples of similarity measures. Section 3 reviews essential data structures. Sections 4 and 5 revisit, generalize, and improve the GREEDY method. Finally we evaluate our proposals in Section 7 and conclude in Section 8.

## 2 Notation and Problem Definition

Let  $\mathcal{D}' = \{d_1, \dots, d_{N-1}\}$  be a collection of  $N-1$  documents. Each  $d_i$  is a string over an alphabet <sup>1</sup>  $\Sigma' = [2, \sigma]$  and is terminated by the sentinel symbol ‘1’, also represent as ‘#’. Adding the one-symbol document  $d_0 = 0$  results in a collection  $\mathcal{D}$  of  $N$  documents. The concatenation  $\mathcal{C} = d_{\pi(0)}d_{\pi(1)} \dots d_{\pi(N-1)}$  is a string over  $\Sigma = [0, \sigma]$ , where  $\pi$  is a permutation of  $[0, N-1]$  with  $\pi(N-1) = 0$ . We denote the length of a document  $d_i$  with  $|d_i| = n_{d_i}$ , and  $|\mathcal{C}| = n$ . See Fig. 1 for a running example. In the “bag of words” model a query  $Q = \{q_0, q_1, \dots, q_{m-1}\}$  is an unordered set of length  $m$ . Each element  $q_i$  is either a *term* (chosen from  $\Sigma'$ ) or a *phrase* (chosen from  $\Sigma'^p$  for  $p > 1$ ). We can now define our problem.

*Top- $k$  document retrieval problem.* Given a collection  $\mathcal{D}$ , a query  $Q$  of length  $m$ , and a similarity measure  $\mathcal{S} : \mathcal{D} \times \Sigma'^m \rightarrow \mathbb{R}$ . Calculate the top- $k$  documents of  $\mathcal{D}$  with regard to  $Q$  and  $\mathcal{S}$ . That is a sorted list of document identifiers  $T = \{\tau_0, \dots, \tau_{k-1}\}$ , with  $\mathcal{S}(d_{\tau_i}, Q) \geq \mathcal{S}(d_{\tau_{i+1}}, Q)$  for  $0 \leq i < k$  and  $\mathcal{S}(d_{\tau_{k-1}}, Q) \geq \mathcal{S}(d_j, Q)$  for  $j \notin T$ .

<sup>1</sup> Note: In Information Retrieval  $\Sigma$  is usually a word alphabet and in String Processing a character alphabet.

$i =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$\mathcal{C}^{word} =$	LA	O	LA	#	O	LA	LA	LA	#	O	O	LA	#	\$
$\mathcal{C} =$	2	3	2	1	3	2	2	2	1	3	3	2	1	0
	$d_1$				$d_3$				$d_2$				$d_0$	

Fig. 1:  $\mathcal{C}$  is the concatenation of a document collection  $\mathcal{D}$  for  $\pi = [1, 3, 2, 0]$ . We use both words (as in  $\mathcal{C}^{word}$ ) or integer identifiers (as in  $\mathcal{C}$ ) to refer to document tokens.

A basic similarity measure used in many self-index based document retrieval systems (see [9]), is the *frequency measure*  $\mathcal{S}^{freq}$ . It scores  $d$  by accumulating the *term frequency* of each term. Term frequency  $f_{d,q}$  is defined as the number of occurrences of term  $q$  in  $d$ ; e.g.  $f_{d_1, \text{LA}} = 2$  in Fig. 1. In IR, more complex  $\text{TF} \times \text{IDF}$  measures also include two additional factors. The first is the inverse of the *document frequency* ( $_{\text{DF}}$ ), which is the number of documents in  $\mathcal{D}$  that contain  $q$ , defined  $F_{\mathcal{D},q}$ ; e.g.  $F_{\mathcal{D}, \text{LA}} = 3$ . The second is the length of the document  $n_d$ . Due to space limitations, we only present the popular Okapi BM25 function:

$$S_{Q,d}^{\text{BM25}} = \sum_{q \in Q} \underbrace{\frac{(k_1 + 1)f_{d,q}}{k_1 \left(1 - b + b \frac{n_d}{n_{\text{avg}}}\right) + f_{d,q}}}_{=w_{d,q}} \cdot \underbrace{f_{Q,q} \cdot \ln \left( \frac{N - F_{\mathcal{D},q} + 0.5}{F_{\mathcal{D},q} + 0.5} \right)}_{=w_{Q,q}} \quad (1)$$

where  $n_{\text{avg}}$  is the mean document length, and  $w_{d,q}$  and  $w_{Q,q}$  refer to components that we address shortly. Parameters  $k_1$  and  $b$  are commonly set to 1.2 and 0.75 respectively. Note that the  $w_{Q,q}$  part is negative for  $F_{\mathcal{D},q} > \frac{N}{2}$ . To avoid negative scores, real-world systems, such as Vigna’s MG4J [1] search engine, set  $w_{Q,q}$  to a small positive value ( $10^{-6}$ ), in this case. We refer to Zobel and Moffat [15] for a survey on IR similarity measures including  $\text{TF} \times \text{IDF}$ , BM25, and LMDS.

### 3 Data Structure Toolbox

We briefly described the two most important building blocks of our systems, and refer the reader to Navarro’s survey [9] and references therein for detailed information. A *wavelet tree* (WT) of a sequence  $X[0, n-1]$  over alphabet  $\Sigma[0, \sigma-1]$  is a perfectly balanced binary tree of height  $h = \lceil \log \sigma \rceil$ , referred to as WT- $X$ . The  $i$ -th node of level  $\ell \in [0, h-1]$  is associated with symbols  $c$  such that  $\lceil c/2^{h-1-\ell} \rceil = i$ . Node  $v$ , corresponding to symbols  $\Sigma_v = [c_b, c_e] \subseteq [0, \sigma-1]$ , represent the subsequence  $X_v$  of  $X$  filtered by symbols in  $\Sigma_v$ . Fig. 2 depicts an example. Only the bitvector which indicates if an element will move to the left or right subtree is stored at each node; that is, WT- $X$  is stored in  $n \lceil \log \sigma \rceil$  bits. Using only sub-linear extra space it is possible to efficiently navigate the tree. Let  $v$  be the  $i$ -th node on level  $\ell < h-1$ , then method  $\text{expand}(v)$  returns in constant time a node pair  $\langle u, w \rangle$ , where  $u$  is the  $(2 \cdot i)$ -th and  $w$  the  $(2 \cdot i + 1)$ -th node on level  $\ell + 1$ . A range  $[l, r] \subseteq [0, n-1]$  in  $X$  can be mapped to range  $[l, r]_v$  in node  $v$  such that the sequence  $X_v[l, r]_v$  represents  $X[l, r]$  filtered by  $\Sigma_v$ . Operation

$expand(v, [l, r]_v)$  then returns in constant time a pair of ranges  $\langle [l, r]_u, [l, r]_w \rangle$  such that the sequence  $X_u[l, r]_u$  (resp.  $X_w[l, r]_w$ ) represents  $X[l, r]$  filtered by  $\Sigma_u$  (resp.  $\Sigma_w$ ). Fig. 2 provides an example.

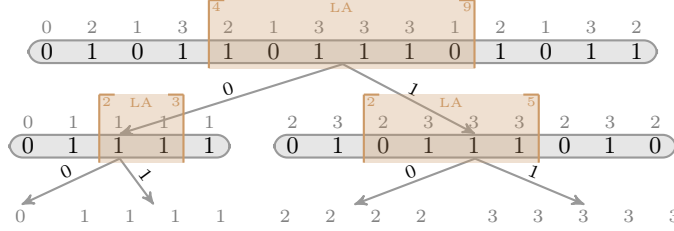


Fig. 2: Wavelet tree over document array  $D$ . Method  $expand(v_{root}, [4, 9])$  maps range  $[4, 9]$  (locus of LA) to range  $[2, 3]$  in the left and range  $[2, 5]$  in the right child.

The *binary suffix tree* (BST) of string  $X[0, n-1]$  is the compact binary trie of all suffixes of  $X$ . For each path  $p$  from the root to a leaf, the concatenation of the edge labels of  $p$ , corresponds to a suffix. The children of a node are ordered lexicographically by their edge labels. Each leaf is labeled with the starting position of its suffix in  $X$ . Read from left to right, the leaves form the *suffix array* ( $SA$ ), which is a permutation of  $[0, n-1]$  such that  $X[SA[i], n-1] <_{lex} X[SA[i+1], n-1]$  for all  $0 \leq i < n-1$ . We refer to Fig. 3 for an example. Compressed versions of  $SA$  and  $ST$  – the compressed  $SA$  ( $CSA$ ) and compressed  $ST$  ( $CST$ ) – use space essentially equivalent to that of the compressed input, while efficiently supporting the same operations. For example, given a pattern  $P$  of length  $m$ , the range  $[l, r]$  in  $SA$  containing all suffixes start with  $P$  or the corresponding node, that is the *locus* of  $P$ , in the BST is calculated in  $\mathcal{O}(m \log \sigma)$ .

## 4 Revisiting and generalizing the GREEDY framework

The GREEDY framework of Culpepper et al. [4] consists of two parts: a  $CSA$  built over  $\mathcal{C}$ , and a  $WT$  over the *document array*  $D[0..N-1]$ ; with document array entry  $D[i]$  specifying the document in which suffix  $SA[i]$  starts. The grey numbers below each  $SA[i]$  value in Fig. 3 correspond to  $D[i]$ . A top- $k$  query using  $\mathcal{S}^{freq}$  with  $m = 1$  is answered as follows. For term  $q = q_0$  the  $CSA$  returns a range  $[l, r]$ , such that all suffixes in  $SA[l, r]$  are prefixed by  $q$ . Note that the size of the range corresponds to  $f_{\mathcal{D}, q}$ , the number of occurrences of  $q$  in  $\mathcal{D}$ . In  $WT-D$  the alphabet  $\Sigma_v$  of each node represents a subset  $\mathcal{D}_v \subseteq \mathcal{D}$  of documents of  $\mathcal{D}$ ; and the size of the mapped interval  $[l, r]_v$  equals  $f_{\mathcal{D}_v, q}$ , the number of occurrences of  $q$  in the sub-collection  $\mathcal{D}_v$ . Each leaf  $v$  in  $WT-D$  corresponds to a document  $d$  in  $\mathcal{D}$ , such that the size of  $[l, r]_v$  equals term frequency  $f_{d, q}$ .

To calculate the documents with maximal  $f_{d, q}$ , i.e. maximizing  $\mathcal{S}_{q, d}^{freq}$ , a max priority queue stores  $\langle v, [l, r]_v \rangle$ -tuples sorted according to interval size. Initially,  $WT-D$ 's root node and  $[l, r]$  is enqueued. The following process is then repeated



### 5.1 Length Estimation by Document Relabelling

We first improve document length estimation in  $\mathcal{D}_v$  by replacing the collection-wide value  $n_{min}$  by the smallest document length  $n_{\bar{d}}$  in the sub-collection  $\mathcal{D}_v$ . The computation of  $n_{\bar{d}}$  can be performed in constant time if the document identifiers are assigned according to the length of the documents. In this case, the smallest document corresponds to smallest symbol in  $\mathcal{D}_v$  which is  $\Sigma_v[0]$ . The latter can be computed in constant time. Let  $v$  be the  $i$ -th node of level  $\ell$  in WT-D<sup>n</sup> then  $\Sigma_v[0] = i \cdot 2^{\lceil \log N \rceil - \ell - 1}$ . The document lengths are maintained in an array  $L[0, N-1]$ . In our example in Fig. 1 and 2 we have reordered the documents using a permutation  $\pi = [1, 3, 2, 0]$ . The additional space of  $N \log N + N \log n_{max}$  bits is negligible compared to the size of the CSA and  $D$ .

### 5.2 Improved term frequency estimation

Unit now we use the range size  $f_{\mathcal{D}_v, q}$  of term  $q$  in  $v$  to estimate an upper bound for the maximal term frequency in a document  $d \in \mathcal{D}_v$ . Knowing the number of distinct documents in  $\mathcal{D}_v$ , called  $F_{\mathcal{D}_v, q}$ , helps to improve the upper bound to the number of repetitions plus one:  $\delta_{\mathcal{D}_v, q} = f_{\mathcal{D}_v, q} - F_{\mathcal{D}_v, q} + 1$ . In this section, we present a method that computes  $\delta_{\mathcal{D}_v, q}$  in constant time during WT-D traversal.

The solution is built on top of Sadakane's [12] document frequency structure (DF), which solves the problem solely for  $\mathcal{C}_v = \mathcal{C}$ . We briefly revisit the structure: first, a BST is built over  $\mathcal{C}$ , see Fig. 3. The leaves are labeled with the corresponding documents, i.e. from left to right  $D$  is formed. The inner nodes are numbered from 1 to  $n-1$  in-order. Each node  $w_i$  holds a list  $\mathcal{R}_i$ , containing all documents which occur in both subtrees of  $w_i$ . We refer to elements in  $\mathcal{R}_i$  as *repetitions*. Let  $w_i$  be the locus of a term  $q$  in the BST and let  $[l, r]$  be  $w_i$ 's interval. Then the total number of repetitions in  $D[l, r]$  can be calculated by accumulating the length of all repetition lists in  $w_i$ 's subtree. To achieve this, Sadakane generated a bitvector  $H$  that concatenates the unary coding of the lengths of all  $\mathcal{R}_i$ :  $H = 10^{|\mathcal{R}_0|}10^{|\mathcal{R}_1|}1 \dots 0^{|\mathcal{R}_{n-1}|}1$ . The subtree interval  $[l, r]$  can be mapped into  $H$  via select operations:  $[l', r'] = [select_1(l, H), select_1(r, H)]$ , since the accumulation of the list lengths equals the number of zeros in  $[l', r']$ . The following example illustrates the process: interval  $[4, 9]$  corresponds to term  $q = \text{LA}$  and is mapped to  $[l', r'] = [select_1(4, H), select_1(9, H)] = [7, 15]$  in  $H$ . It follows that there are  $z_l = l' - l = 3$  zeros in  $H[0, l']$  and  $z_r = r' - r = 6$  in  $H[0, r']$ ; thus there are  $6 - 3 = 3$  repetitions in  $D[4, 9]$ . We can overestimate the maximal term frequency by assuming that all repetitions belong to the same document  $d_x$  and add one for  $d_x$  itself. So  $\delta_{\mathcal{D}_v, q} = 4$  in this case. This overestimates the maximal term frequency, which is  $f_{d_3, q} = 3$ , by one. The interval size estimate would have been 6

We now extend Sadakane's solution to work on all subsets  $\mathcal{D}_v$ . First, we concatenate all  $\mathcal{R}_i$  and form the *repetition array*  $R[0, n-N-1]$  (again, see Fig. 3), containing the actual repetition value for each zero in  $H$ . As above, using  $H$  and  $select_1$ , we can map  $[l, r]$  to the corresponding range  $[l'', r''] = [z_l, z_r - 1]$  in  $R$ . To calculate  $\delta_{\mathcal{D}_v, q}$  for  $\mathcal{D}_v$  we represent  $R$  as a WT. Now, we can traverse

WT-D and WT- $R$  simultaneously, mapping  $[l'', r'']$  to  $[l'', r'']_v$  in WT- $R$ . The size of  $[l'', r'']_v + 1$  equals  $\delta_{\mathcal{D}_v, q}$  since node  $v$  contains only repetitions of  $\mathcal{D}_v$ .

## 6 Space reduction

The space of  $R$  can be reduced to array  $\hat{R}$  by omitting all elements belonging to the root  $v_{ST}$  of the non-binary ST since we will never query the empty string. In Fig. 3 all nodes with empty path labels correspond to  $v_{ST}$ , i.e.  $v_1, v_4$ , and  $v_{10}$ . Hence  $\hat{R} = \{3, 3, 1, 2\}$  and we employ a bitvector to map from the index domain of  $R$  into  $\hat{R}$ .

Second, we note that the space of WT-D and WT- $\hat{R}$  can be reduced if the length of query phrase is restricted to length  $\ell$ . In this case, we can sort ranges in  $\hat{R}$  which belong to nodes  $v_i$ , where  $v_i$  are the loci of patterns of length  $\ell$ . Since all query ranges are aligned at borders of sorted ranges, the interval sizes during processing will not be affected. In our running example, if  $\ell = 1$ , we can sort the elements of  $v_9$ 's subtree, resulting in  $\hat{R}^1 = \{1, 3, 3, 2\}$ . The sorting will result in better compressibility of WT- $\hat{R}^\ell$ .

Third, we observe that when using WT- $\hat{R}^\ell$  *only a part of WT-D* is necessary to calculate  $\delta_{\mathcal{D}_v, q}$ . If  $q$  occurs more than once in  $\mathcal{D}_v$ , WT- $\hat{R}^\ell$  can be used to get  $\delta_{\mathcal{D}_v, q}$ . Hence, WT-D is only used to determine the existence of  $q$  in  $\mathcal{D}_v$ , and we only store need to store the unique values inside ranges corresponding to loci of  $\ell$ -length patterns. In addition, values in these ranges can be sorted, since this does not change the result of the existence queries. In our example we get  $D^1 = \{3, 0, 1, 2, 0, 1, 2, 0, 1, 2\}$ ; one increasing sequence per symbol. A bitvector is again used to map into  $D^\ell$ .

## 7 Experimental Study

*Indexes and Implementations.* To evaluate our proposals we created the SUccinct Retrieval Framework (SURF)<sup>2</sup> which implements document retrieval specific components, like Sadakane's DF structure. These components can be parametrized by structures provided by the SDSL library [6]. We assembled three self-index based systems, corresponding to different functionality-space trade-offs. All systems use the same CSA and DF structure. The CSA is implemented as an FM-index using a WT. This WT as well as DF use a practical compressed bitvector [11] to minimize space, since all query related operations on these components take only a fraction of a millisecond.

Our first index (I-D<sup>n</sup>) adds WT-D<sup>n</sup>, which uses an uncompressed bitvector to allow fast WT traversal. Our second structure (I-D<sup>n</sup> $\hat{R}^n$ ) adds WT- $\hat{R}^n$ . It uses a compressed bitvector to compress the increasing sequences in  $\hat{R}^n$ . To show a functionality-space trade-off we also provide the previous index with a phrase length restriction of one, named I-D<sup>1</sup> $\hat{R}^1$ . The components WT-D<sup>1</sup> and WT- $\hat{R}^1$  both use compressed bitvectors to minimize the space of the WTs.

<sup>2</sup> Source code, test queries, and the scripts to reproduce the experiments are publicly available at <https://github.com/simongog/surf/>.

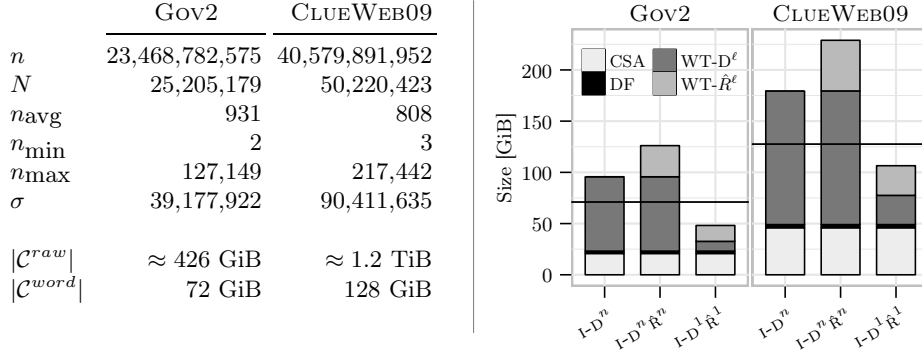


Fig. 4: Collection statistics for Gov2 and CLUEWEB09 (left) and memory breakdown of our indexes (right).  $|\mathcal{C}^{raw}|$  denotes the original size of the collection, while  $|\mathcal{C}^{word}|$  denotes the size after parsing it. A more detailed space breakdown of the indexes is available at <http://go.unimelb.edu.au/6a4n>.

As a reference point we also implemented a competitive inverted file based search index (INVIDX) which stores block-based postings lists compressed using OPTPFD [8,14]. For each block, a representative is stored to allow efficient skipping. The document ranking is calculated using two list processing schemes. The first scheme – INVIDX-W – uses the efficient WAND list processing algorithm [2]. However, WAND and other advanced processing schemes require similarity measure specific pre-computation during construction time. A more flexible, but less efficient processing scheme – INVIDX-E – exhaustively evaluates all postings in document-at-a-time order without either the burden or benefit of score pre-computation.

*Data Sets, Queries and Test Environment.* We compare our index structures over two standard IR test collections: GOV2 and CLUEWEB09. GOV2 is the test collection of the TREC 2004 Terabyte Track competition and the CLUEWEB09 collection consists of the English text “Category B” subset of the ClueWeb09 dataset<sup>3</sup>. To ensure reproducibility we extract the integer token sequence  $\mathcal{C}$  for both collections from Indri<sup>4</sup> using default parameters. No stop-words were removed from the collection. We evaluate our indexes using queries chosen from the TREC 2005 and TREC 2006 Terabyte track efficiency queries<sup>5</sup>. A total of 1000 queries were randomly sampled from both query sets, ensuring all query terms are present in both test collections. Statistics for both collections, given in Fig 4 (left), are in line with other studies [13]. We support ranked disjunctive (Ranked-OR, at least one term must occur) and ranked conjunctive (Ranked-AND, all terms must occur) retrieval. All indexes were implemented using C++11 and compiled using GCC 4.8.1 with optimizations. Our machine was equipped with 256 GiB RAM and we used one Intel Sandybridge core (E5-2680) running at 2.7 Ghz. All indexes were loaded into main memory prior to query processing.

<sup>3</sup> <http://lemurproject.org/clueweb09/>

<sup>4</sup> <http://www.lemurproject.org/indri/>

<sup>5</sup> <http://trec.nist.gov/data/terabyte.html>



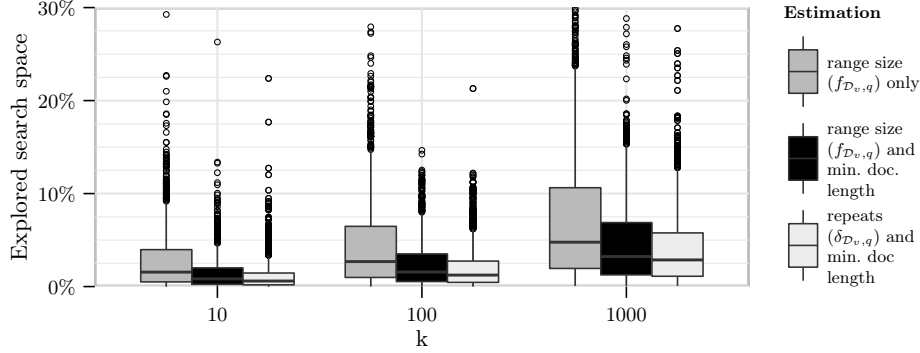


Fig. 5: Percentage of states evaluated  $k = 10, 100$ , and  $1000$  during Ranked-OR retrieval using BM25 for both TREC2005 and TREC2006 query sets for Gov2.

*Space Usage.* The space usage of our indexes is summarized in Fig. 4 (right). Note that our smallest index is 4 to 5 times larger than our reference inverted index. However, an inverted index supporting phrase queries would require additional positional information, which would significantly increase its size. The size of our integer parsing of size  $n \lceil \log \sigma \rceil$  is shown as a horizontal line. The CSA for both collections compresses to roughly 30% of the size of the integer parsing. The space for DF is negligible. The WT- $D^n$  has the size of the integer parsing plus 5% overhead for a rank structure. The size reduction from  $R$  to  $\hat{R}$  is substantial. For example, storing  $R$  for CLUEWEB09 requires 123 GiB, whereas  $\hat{R}$  requires only 74 GiB. Restricting the phrase length to one ( $I-D^1\hat{R}^1$ ), which makes it equivalent to a non-positional inverted index, shrinks the structure below the original input size.

*Processed States.* In the first experiment, we measure the quantitative effects of our improved score estimation during GREEDY processing. We compare the range size ( $f_{D_{v,q}}$ )-only estimation to (a) range size estimation including document length estimation and (b) repeats estimation ( $\delta_{D_{v,q}}$ ) including document length estimation. Fig. 5 shows the percentage of processed states for all methods and  $k = \{10, 100, 1000\}$  for both query sets on GOV2 using BM25 Ranked-OR processing. The percentage is calculated as the fraction of states processed compared to the exhaustive processing of each query ( $k = N$ ). For all  $k$ , range size only estimation evaluates the most states on average. For  $k = 10$ , the median percentage of evaluated states for range size only estimation is 1.6%. Adding document length estimation reduces the number of evaluated states by half to 0.8%. Using  $\delta_{D_{v,q}}$  instead of  $f_{D_{v,q}}$  to estimate the frequency further improved the percentage of evaluated states to 0.06%. Similar effects can be observed for  $k = 100$  and  $k = 1000$ . For  $k = 1000$ , document length estimation reduces the percentage from 5.1% to 3.2%. Frequency estimation using  $\delta_{D_{v,q}}$  again marginally improves the number of evaluated nodes to 2.8%. Overall, document length estimation has a larger impact on GREEDY than better frequency estimation via  $\delta_{D_{v,q}}$ .

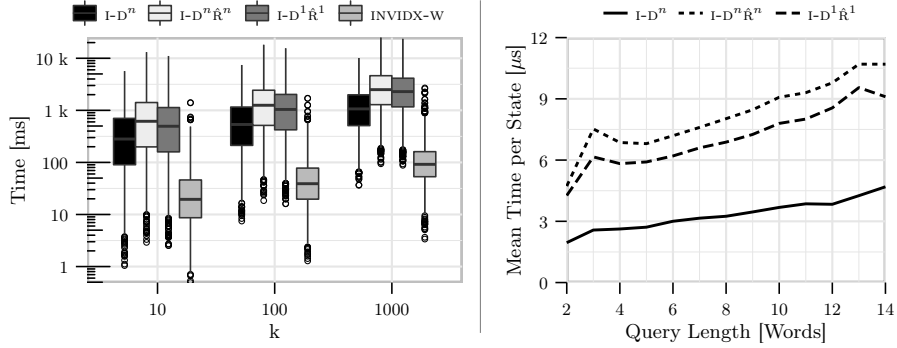


Fig. 6: Runtime in milliseconds (left) and time to process one WT state for  $k = 100$  (right) for BM25 Ranked-OR retrieval on Gov2.

*Disjunctive Ranked Retrieval.* Next we evaluate the runtime performance of our indexes  $I-D^n$ ,  $I-D^n \hat{R}^n$ ,  $I-D^1 \hat{R}^1$  for BM25 Ranked-OR query processing. Fig. 6 (left) shows runtime on Gov2 and both query sets for  $k = \{10, 100, 1000\}$ . We additionally included INVIDX-W as a reference point for an efficient inverted index. The latter uses additional similarity measure dependent information and clearly outperforms all self-index based indexes. For  $k = 10$ , it achieves a median runtime performance of less than 20 milliseconds, and performs well for other test cases. Our fastest index,  $I-D^n$ , is roughly 15 times slower, achieving a median runtime of 300 milliseconds for  $k = 10$ . The other two indexes,  $I-D^n \hat{R}^n$  and  $I-D^1 \hat{R}^1$  are approximately two times slower than  $I-D^n$ . This can be explained by the fact that  $I-D^n$  uses an uncompressed WT, whereas the other indexes use compressed WTs to save space. Also note that  $I-D^1 \hat{R}^1$  is faster than  $I-D^n \hat{R}^n$  as ranges in  $\hat{R}^1$  can be sorted, which creates runs in the WT which in turn allows faster state processing. The mean time per processed state – depicted in Fig. 6 (right) – highlights this observation. For  $I-D^n$ , the time linearly increases from 2 to 5 microseconds. While there is a correlation to the number of query terms, rank operations occur in close proximity – cache friendly – within  $WT-D^n$ , which increases performance. For the other indexes, we simultaneously access two WTs at different locations to evaluate a single state. This reduces caching effects resulting in a stronger dependence on query length.

*Efficient Retrieval using Multi-Word Expressions.* Next we demonstrate one example of additional functionality provided by our self-indexed based systems. We use the concept of *strong associativity* [3] which defines the ratio of joint probability against the probabilities of a random co-occurrence as an indicator of a multi word expression (MWE). We create MWEs on-the-fly during query time using the text statistics provided by the CSA. We use a simple scheme which greedily parses each query into MWEs. For example, instead of processing the terms “saudi” and “arabia” independently, we instead process the MWE “saudi arabia” as a single query term. This can significantly reduce the query time of our indexing schemes. We explore the efficiency of such a “phrase” processing scheme

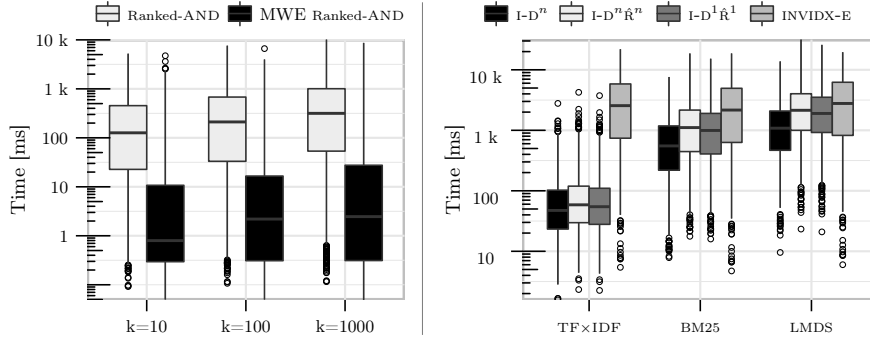


Fig. 7: Ranked-AND BM25 runtime for unparsed and MWE-parsed queries (right)  
Ranked-OR runtime for different similarity measures and indexes.

in Fig. 7 (left). The figure shows the runtime for queries from the TREC2006 query sets for GOV2 using  $I-D^n$ . For all  $k$ , the runtime is reduced by an order of magnitude. This experiment shows how our system would support retrieval tasks where the vocabulary does not consist of words but a large number of entities, since MWE capture the latter. Supporting MWE does not increase the size of our index, but would substantially increase the size of an inverted index.

*Flexible Ranked Document Retrieval.* Another virtue of our proposal is scoring flexibility. The indexes efficiently support a wide range of similarity measures, which can be changed and tuned after the index is built, while optimized inverted indexes require similarity measure-dependent pre-computation during construction [2]. If ranking functions are only chosen at query time, inverted indexes exhaustively process postings lists to retrieve the top- $k$  documents. To highlight the benefit of flexibility, we compare our index structures to INVIDX-E using three different ranking formulas:  $TF \times IDF$ , BM25, and LMDS. Fig. 7 (right) shows the results on GOV2. Interestingly, our index structures significantly outperform the exhaustive inverted index for  $TF \times IDF$ . This can be attributed to the large influence of the document length  $n_d$  on  $\mathcal{S}^{TF \times IDF}$ . Unlike BM25 or LMDS, the final document score is linearly proportional to the actual size of the document, thus document length estimation significantly reduces the number of evaluated states. For BM25, the document length contribution to the final document score is scaled in reference to the average document length, and thus has a smaller effect on the overall score of each document. While our indexes still outperform INVIDX-E, the difference is less significant than for  $TF \times IDF$ .

## 8 Conclusions and Future Work

We have presented a versatile self-index based retrieval framework which allows rank-safe top- $k$  retrieval on multi-term queries using complex scoring functions. The proposed estimation methods have substantially improved the query speed compared to frequency-only score estimation. In our experiments we found that

top- $k$  document retrieval is still solved more efficiently by inverted indexes, if augmented by similarity measure-dependent pre-computations. However, self-index based systems provide more functionality and thus can be used in scenarios where the inverted index is not applicable or slower. We believe that GREEDY performance can be further improved, e.g. by incorporating range majority queries into the score estimation. We provide our framework as open-source to enable researchers of different research disciplines to profit from the functionality provided by self-index based search systems.

## Acknowledgments

We are grateful to Paul Cook, who pointed us to [3], and Alistair Moffat and Andrew Turpin for fixing our grammar. This research was supported by a Victorian Life Sciences Computation Initiative (VLSCI) grant number VR0052 on its Peak Computing Facility at the University of Melbourne, an initiative of the Victorian Government. Both authors were funded by ARC DP grant DP110101743.

## References

1. Boldi, P., Vigna, S.: MG4J at TREC 2005. In: Proc. TREC (2005)
2. Broder, A.Z., Carmel, D., Herscovici, H., Soffer, A., Zien, J.: Efficient query evaluation using a two-level retrieval process. In: Proc. CIKM. pp. 426–434 (2003)
3. Church, K.W., Hanks, P.: Word association norms, mutual information, and lexicography. *Computational Linguistics* 16(1), 22–29 (1990)
4. Culpepper, J.S., Navarro, G., Puglisi, S.J., Turpin, A.: Top- $k$  ranked document search in general text databases. In: Proc. ESA. pp. 194–205 (2010)
5. Culpepper, J.S., Petri, M., Scholer, F.: Efficient in-memory top- $k$  document retrieval. In: Proc. SIGIR. pp. 225–234 (2012)
6. Gog, S., Beller, T., Moffat, A., Petri, M.: From theory to practice: Plug and play with succinct data structures. In: Proc. SEA (2014), to appear.
7. Konow, R., Navarro, G.: Faster compact top- $k$  document retrieval. In: Proc. DCC. pp. 351–360 (2013)
8. Lemire, D., Boytsov, L.: Decoding billions of integers per second through vectorization. *Soft. Prac. & Exp.* (2013), doi:10.1002/spe.2203
9. Navarro, G.: Spaces, trees and colors: The algorithmic landscape of document retrieval on sequences. *ACM Computing Surveys* 46(4) (2014)
10. Navarro, G., Nekrich, Y.: Top- $k$  document retrieval in optimal time and linear space. In: Proc. SODA. pp. 1066–1077 (2012)
11. Navarro, G., Provel, E.: Fast, small, simple rank/select on bitmaps. In: Proc. SEA. pp. 295–306 (2012)
12. Sadakane, K.: Succinct data structures for flexible text retrieval systems. *J. Discrete Alg.* 5(1), 12–22 (2007)
13. Vigna, S.: Quasi-succinct indices. In: Proc. WSDM. pp. 83–92 (2013)
14. Yan, H., Ding, S., Suel, T.: Inverted index compression and query processing with optimized document ordering. In: Proc. WWW. pp. 401–410 (2009)
15. Zobel, J., Moffat, A.: Inverted files for text search engines. *ACM Comp. Surv.* 38(2) (2006)

## A Correctness of GREEDY

**Lemma 1.** *Given a document collection  $\mathcal{D}$ , a query  $Q$ , and a similarity measure  $\mathcal{S}$ . Algorithm GREEDY reports the top- $k$  documents for  $Q$  of  $\mathcal{D}$  if*

- *at every node  $v$  the score estimate  $s_v$  is not smaller than the maximum document score in its subtree*
- *and the score estimates  $s_u$  and  $s_w$  of  $v$ 's children is never larger than  $s_v$ .*

*Proof.* Assume that there is a unreported document  $d'$  which has a score larger than the  $k$ -th reported document  $d$ ; i.e.  $s_{v'} > s_v$ . Then  $d'$  was not in the queue when  $d$  was reported, since otherwise  $d'$  would have been reported first. Therefore an ancestor  $d''$  of  $d'$  has to be in the queue. This is not possible since when  $d$  was extracted all score estimates in the queue were smaller or equal to the score estimate  $s_v$  of  $d$ . But the score estimate  $s_{v''}$  of  $d''$  is larger or equal then  $s_{v'}$  and hence larger than  $s_v$ . This is a contradiction.  $\square$

## B Additional Similarity Measures

A simple TF $\times$ IDF formulation given in the survey paper of Zobel and Moffat [15]:

$$S_{Q,d}^{\text{TF}\times\text{IDF}} = \frac{1}{n_d} \sum_{q \in Q} \underbrace{(1 + \ln f_{d,q})}_{=w_{d,q}} \cdot \underbrace{\ln \left( 1 + \frac{N}{F_{\mathcal{D},q}} \right)}_{=w_{Q,q}} \quad (2)$$

Another similarity function is based on an Language Model (LMDS) formulation:

$$S_{Q,d}^{\text{LM}} = m \cdot \ln \left( \frac{\mu}{n_d + \mu} \right) + \sum_{q \in Q} \underbrace{\ln \left( \frac{f_{d,q}}{\mu} \cdot \frac{n}{F_{\mathcal{D},q}} + 1 \right)}_{=w_{d,q}} \cdot \underbrace{f_{Q,q}}_{=w_{Q,q}} \quad (3)$$

Parameter  $\mu$  is typically set to 2,500.

## C Collection Statistics and Experimental Results.

Examples of multi-word expressions (MWEs) detected using strong associativity and text statistics provided by the CSA. Each detected MWE is shown in brackets.

- |  |                                     |
|--|-------------------------------------|
| – (fort myers florida) (blue crown     | – (first tennessee bank) (web site) |
| conure) (bird breeder)                 | – (are hot dogs) (healthy food)     |
| – map of (saudi arabia)                | – firex (smoke alarm) (downer grove |
| – the (sisterhood of the travel pants) | illinois)                           |
| movie                                  | – (bayside raider) queens (football |
| – (h1 b visa)                          | coach)                              |

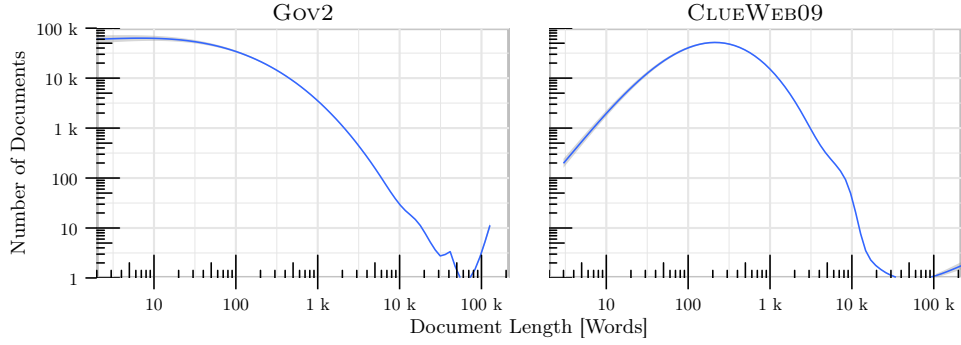


Fig. 8: Document lengths distribution for Gov2 and CLUEWEB09.

Document statistics for both test collections are shown in Figure 8. During the creation of each collection an upper limit was defined to limit the size of documents.

Figure 9 shows results for Ranked-AND and Ranked-OR retrieval over the larger CLUEWEB09 collection with similar results to that shown for Gov2 above.

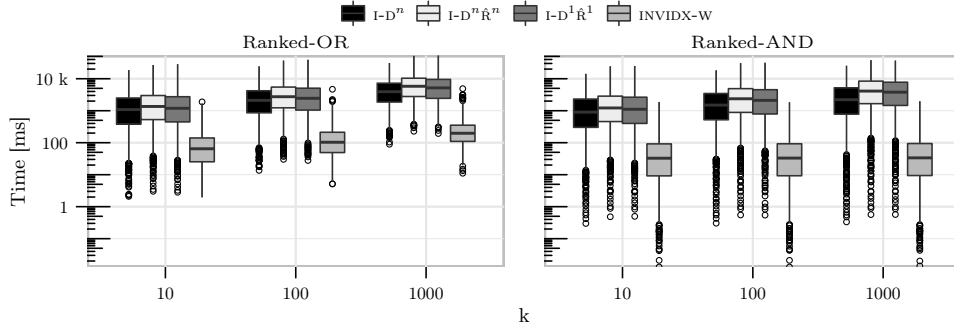


Fig. 9: Disjunctive (Ranked-OR) and conjunctive (Ranked-AND) top- $k$  query time for  $k = 10, 100$  and  $1000$  for CLUEWEB09 and query sets and BM25 in milliseconds.